

Dynamic Graph Queries*

Pablo Muñoz^{†1}, Nils Vortmeier², and Thomas Zeume^{‡2}

- 1 Department of Computer Science, University of Chile
CIWS Center for Semantic Web Research
Chile
`pmunoz@dcc.uchile.cl`
- 2 TU Dortmund University
Germany
`{nils.vortmeier, thomas.zeume}@tu-dortmund.de`

Abstract

Graph databases in many applications—semantic web, transport or biological networks among others—are not only large, but also frequently modified. Evaluating graph queries in this dynamic context is a challenging task, as those queries often combine first-order and navigational features.

Motivated by recent results on maintaining dynamic reachability, we study the dynamic evaluation of traditional query languages for graphs in the descriptive complexity framework. Our focus is on maintaining regular path queries, and extensions thereof, by first-order formulas. In particular we are interested in path queries defined by non-regular languages and in extended conjunctive regular path queries (which allow to compare labels of paths based on word relations). Further we study the closely related problems of maintaining distances in graphs and reachability in product graphs.

In this preliminary study we obtain upper bounds for those problems in restricted settings, such as undirected and acyclic graphs, or under insertions only, and negative results regarding quantifier-free update formulas. In addition we point out interesting directions for further research.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Dynamic descriptive complexity, graph databases, graph products, reachability, path queries

Digital Object Identifier 10.4230/LIPICs.xxx.yyy.p

1 Introduction

Graph databases are important in applications in which the topology of data is as important as the data itself. Intuitively, a graph database represents objects (by nodes), and relationships between those objects (often modeled by labeled edges—see [1] for a survey on graph database models). The last years have witnessed an increasing interest in graph databases, due to the uprise of applications that need to manage and query massive and highly-connected data, as for example the semantic web, social networks or biological networks. In most of these applications, databases are not only large, but also highly dynamic. Data is frequently

* Parts of this work are also included in the dissertation thesis of the third author [23].

[†] The author acknowledges the financial support by Conicyt PhD scholarship and Millennium Nucleus Center for Semantic Web Research under Grant NC120004.

[‡] The author acknowledges the financial support by DFG grant SCHW 678/6-1.



© Pablo Muñoz, Nils Vortmeier and Thomas Zeume;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

inserted and deleted, and hence so is its network structure. The goal of this work is to explore how query languages for graph databases can be evaluated in this dynamic context.

Many query languages for graph databases combine traditional first-order features with *navigational* ones. Already basic languages (such as regular path queries, see e.g. [21, 2]) allow to test the existence of paths satisfying constraints on their labels (e.g. adherence to a regular expression in regular path queries). Computing the answers to this kind of queries on large, highly dynamic graphs is a big challenge. It is conceivable, though, for answers to a query before and after small modifications to be closely related. Thus a reasonable hope is to be able to update the answer to a query in a more efficient way than recomputing it from scratch after each modification. Even more so if we allow to store extra auxiliary data that might ease the updating task. To what extent this is possible, and in which precise conditions, is the subject of *dynamic computational complexity*.

Here we are interested in studying the dynamic complexity of query languages for graph databases from a *descriptive* approach. In the dynamic descriptive complexity setting, proposed independently by Dong, Su and Topor [9, 8] and by Patnaik and Immerman [16], a *dynamic program* maintains auxiliary relations with the intention to help answering a query over a (relational) database subject to small modifications (insertions or deletions of tuples). When a modification occurs, the query answers and every auxiliary relation are updated by first-order formulas (or, equivalently, by core SQL queries) evaluated over the current database and the available auxiliary data. Such programs benefit therefore from being both highly parallelizable (due to the close connection of first-order logic and small depth boolean circuits) and readily implementable in standard relational database engines. The class of queries maintainable by first-order update formulas is called DYNFO.

Query languages for graphs have, so far, not been studied systematically in the dynamic descriptive complexity setting. Very likely the main reason is that until recently it was not even known whether reachability in directed graphs could be maintained by first-order update formulas. That this indeed is possible was shown in [6], with the immediate consequence that all fixed (conjunctions of) regular path queries can also be maintained. Thus regular path queries can be evaluated in a highly parallel fashion in dynamic graph databases.

Motivated by this result we study the dynamic maintainability of more expressive query languages.

► **Goal.** *Gain a better understanding of the limits of maintaining graph query languages in the dynamic context.*

Our focus is on regular path queries and extensions thereof—non-regular path queries and extended conjunctive regular path queries (short: ECRPQs).

Some previous work on non-regular path queries has been done. Weber and Schwenck exhibited a context-free path query (the Dyck language D_2) that can be maintained in DYNFO on acyclic graphs [20]. Also, for the simple class of path-shaped graph databases, formal language results can be transferred. Already Patnaik and Immerman pointed out that regular languages can be maintained in DYNFO [16]. Later, Gelade et al. systematically studied the dynamic complexity of formal languages [11]. They showed, among other results, that regular languages can be maintained by quantifier-free update formulas, and that all context-free languages can be maintained in DYNFO.

The second extension of regular path queries to be studied here are extended conjunctive regular path queries. In previous work it has been noticed that conjunctions of regular path queries (CRPQs) fall short in expressive power for modern applications of graph databases [4]. A feature commonly demanded by these applications is the comparison of

labels of paths defined by CRPQs based on relations of words (e.g. prefix, length constraints, fixed edit-distance). ECRPQs have been introduced to fulfill this requirement [4], that is, they generalize CRPQs by allowing to test whether multiple labels of paths adhere to given regular relations. Two basic properties expressible by ECRPQs are whether two pairs of nodes are connected by paths of the same length and if so, whether also paths with the same label sequence exist. In general, maintaining the result of ECRPQs seems to be a difficult task. In this article we therefore explore the maintenance of ECRPQs in restricted settings.

Finally, there is also a close connection between the evaluation of graph queries and the reachability problem in unlabeled and labeled product graphs. We discuss this connection (see Section 2), and exploit it in several of our results.

Contributions First we study path queries and show that

- all regular path queries can be maintained by quantifier-free formulas when only insertions are allowed,
- all context-free path queries can be maintained by first-order formulas on acyclic graphs, and
- there are non-context-free path queries maintainable by first-order formulas on undirected and acyclic graphs, as well as on general graphs under insertions only.

As a first step towards maintaining ECRPQs we explore for which graph classes the lengths of paths between nodes can be maintained. We exhibit dynamic programs for maintaining all distances for undirected and acyclic graphs, as well as for directed graphs when only insertions are allowed. It remains open, whether distances can be maintained in DYNFO for general directed graphs, but we show that quantifier-free update formulas do not suffice.

The techniques used to maintain all distances can be used to maintain variants of ECRPQs in restricted settings. Denote the extension of a class of queries by linear constraints on the number of occurrences of symbols on paths by +LC. This extension was introduced and studied in [4]. We show that

- all CRPQ+LCs can be maintained by first-order formulas when only insertions are allowed, and
- all ECRPQ+LCs can be maintained by first-order formulas on acyclic graphs.

An immediate consequence of our results for distances is that reachability can be maintained in products of (unlabeled) graphs for those restrictions. By using the dynamic program for maintaining the rank of matrices from [6], we extend this result to more general graph products. Furthermore we show that pairs of nodes connected by paths with the same label sequence can be maintained in acyclic graphs using first-order update formulas.

Related work The maintenance of problems has also been studied from an algorithmic point of view. A good starting point for readers interested in upper bounds for dynamic algorithms is [18, 7]; a good starting point for lower bound techniques is the survey by Miltersen on cell probe complexity [14]. The upper bounds for reachability obtained in [18, 7] immediately transfer to dynamic algorithmic evaluation of regular path queries (using the reduction exhibited in [6]).

Outline The dynamic setting and the basic graph query languages are introduced in Section 2. There we also discuss the connection between query evaluation and reachability in product graphs. Section 3 contains the results on maintaining graph queries. Our results for maintaining distances and ECRPQs are presented in Section 4. In Section 5 some of the results for maintaining graph queries are transferred to reachability in graph products, and we also provide results for reachability in generalized graph products. We conclude in Section 6.

This is a full version of [15].

Acknowledgements We thank Pablo Barceló, Samir Datta and Thomas Schwentick for stimulating and illuminating discussions.

2 Preliminaries

In this section we introduce the dynamic complexity framework as well as the graph query languages used in this article.

Dynamic complexity framework In this work we use the dynamic complexity framework as introduced by Patnaik and Immerman [16]. The following introduction of the framework is borrowed from previous work [25].

Intuitively, the goal of a dynamic program is to keep the result of a given query Q up to date while the database to be queried (the *input database*) is subject to tuple insertions and deletions. To this end the dynamic program stores auxiliary relations (the *auxiliary database*) with the aim that one of those relations always (that is, after every possible sequence of modifications), stores the result of Q for the current input structure. Whenever a tuple is inserted into or deleted from the input structure, each auxiliary relation is updated by the dynamic program by evaluating a specified first-order formula.

We make this more precise now. A *dynamic instance* of a query Q is a pair (\mathcal{D}, α) , where \mathcal{D} is a database over some finite domain D and α is a sequence of modifications to \mathcal{D} . Here, a *modification* is either an insertion of a tuple over D into a relation of \mathcal{D} or a deletion of a tuple from a relation of \mathcal{D} . The result of Q for (\mathcal{D}, α) is the relation that is obtained by first applying the modifications from α to \mathcal{D} and then evaluating Q on the resulting database. We use the Greek letters α and β to denote modifications as well as modification sequences. The database resulting from applying a modification α to a database \mathcal{D} is denoted by $\alpha(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of modifications $\alpha \stackrel{\text{def}}{=} \alpha_1 \dots \alpha_m$ to a database \mathcal{D} is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \alpha_m(\dots(\alpha_1(\mathcal{D}))\dots)$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every (input) database \mathcal{D} , an initial state with initial auxiliary data. The latter defines how the new state of the dynamic program is obtained from the current state when applying a modification.

A *dynamic schema* is a tuple $(\tau_{\text{in}}, \tau_{\text{aux}})$ where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary database, respectively. While τ_{in} may contain constants, we do not allow constants in τ_{aux} in the basic setting. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}}$.

► **Definition 1 (Update program).** An *update program* P over a dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a set of first-order formulas (called *update formulas* in the following) that contains, for every relation symbol R in τ_{aux} and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ with $S \in \tau_{\text{in}}$, an update formula $\phi_\delta^R(\bar{x}; \bar{y})$ over the schema τ where \bar{x} and \bar{y} have the same arity as S and R , respectively.

A *program state* \mathcal{S} over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a structure $(D, \mathcal{I}, \mathcal{A})$ where D is a finite domain, \mathcal{I} is a database over the input schema (the *current database*) and \mathcal{A} is a database over the auxiliary schema (the *auxiliary database*).

The semantics of update programs is as follows. Let P be an update program, $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ be a program state and $\alpha = \delta(\bar{a})$ a modification where \bar{a} is a tuple over D and $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ for some $S \in \tau_{\text{in}}$. If P is in state \mathcal{S} then the application of α yields the new state $\mathcal{P}_\alpha(\mathcal{S}) \stackrel{\text{def}}{=} (D, \alpha(\mathcal{I}), \mathcal{A}')$ where, in \mathcal{A}' , a relation symbol $R \in \tau_{\text{aux}}$ is interpreted by $\{\bar{b} \mid \mathcal{S} \models \phi_\delta^R(\bar{a}; \bar{b})\}$. The effect $P_\alpha(\mathcal{S})$ of applying a modification sequence $\alpha \stackrel{\text{def}}{=} \alpha_1 \dots \alpha_m$ to a state \mathcal{S} is the state $P_{\alpha_m}(\dots(P_{\alpha_1}(\mathcal{S}))\dots)$.

► **Definition 2** (Dynamic program). A *dynamic program* is a triple (P, INIT, Q) , where

- P is an update program over some dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$,
- INIT is a mapping that maps τ_{in} -databases to τ_{aux} -databases, and
- $Q \in \tau_{\text{aux}}$ is a designated *query symbol*.

A dynamic program $\mathcal{P} = (P, \text{INIT}, Q)$ *maintains* a query Q if, for every dynamic instance (\mathcal{D}, α) , the query result $\mathcal{Q}(\alpha(\mathcal{D}))$ coincides with the content of Q in the state $\mathcal{S} = P_\alpha(\mathcal{S}_{\text{INIT}}(\mathcal{D}))$ where $\mathcal{S}_{\text{INIT}}(\mathcal{D})$ is the initial state for \mathcal{D} , that is, $\mathcal{S}_{\text{INIT}}(\mathcal{D}) \stackrel{\text{def}}{=} (D, \mathcal{D}, \text{INIT}(\mathcal{D}))$.

The following example due to [16] shows how the transitive closure of an acyclic graph subject to edge insertions and deletions can be maintained in this set-up. The basic technique of this example will be crucial in some of the later proofs.

► **Example 3.** Consider an acyclic graph G subject to edge insertions and deletions. In the following, our goal is to maintain the transitive closure of G using a dynamic program with first-order update formulas. It turns out that if the graph is guaranteed to remain acyclic, then it is sufficient to store the current transitive closure relation in an auxiliary relation T . We follow the argument from [16].

When an edge (u, v) is inserted into G the following very simple rule updates T : there is a path from x to y after inserting (u, v) if (1) there was already a path from x to y before the insertion, or (2) there were paths from x to u and from v to y before the insertion. This rule can be easily specified by a first-order update formula that defines the updated transitive closure relation¹: $\phi_{\text{INS}_E}^T(u, v; x, y) \stackrel{\text{def}}{=} T(x, y) \vee (T(x, u) \wedge T(v, y))$.

Deletions are slightly more involved. There is a path ρ from x to y after deleting an edge (u, v) if there was a path from x to y before the deletion and (1) there was no such path via (u, v) , or (2) there is an edge (z, z') on ρ such that u can be reached from z but not from z' . If there is still a path ρ from x to y , such an edge (z, z') must exist, as otherwise u would be reachable from y , contradicting acyclicity. This rule can be described by a first-order formula:

$$\begin{aligned} \phi_{\text{DEL}_E}^T(u, v; x, y) \stackrel{\text{def}}{=} & T(x, y) \wedge \left((\neg T(x, u) \vee \neg T(v, y)) \vee \exists z \exists z' \right. \\ & \left. (T(x, z) \wedge E(z, z') \wedge (z \neq u \vee z' \neq v) \wedge T(z', y) \wedge T(z, u) \wedge \neg T(z', u)) \right) \end{aligned}$$

◀

A word on the initial input and auxiliary databases is due. As default we use the original setting of Patnaik and Immerman, where the input database is empty at the beginning, and the auxiliary relations are initialized by first-order formulas evaluated on the initial input

¹ For simplicity we use the same names for elements and variables.

database. When we use a different initialization setting we state it explicitly. In the literature several other settings have been investigated and we refer to [25, 23] for a detailed discussion.

The class of queries that can be maintained by first-order update formulas in the setting of Patnaik and Immerman is called² DYNFO. Restricting update formulas to be quantifier-free yields the class DYNPROP.

When showing that a particular query is in DYNFO we often assume that *arithmetic* on the domain is available from initialization time, that is, we assume the presence of relations $\leq, +, \times$ that are interpreted as a linear order—allowing to identify elements with numbers—, addition and multiplication on the domain. From a DYNFO program that relies on built-in arithmetic, a program without built-in arithmetic can be constructed for all queries studied here by using a technique from [6].

► **Proposition 4** ([6, Theorem 4]). *Every domain-independent query Q that can be maintained in DYNFO with built-in arithmetic can also be maintained in DYNFO.*

Here, a query is *domain-independent* if its result does not change when elements are added to the domain.

Constructing a DYNFO program for a specific query Q can be a tedious task. Such a construction can often be simplified by reducing Q to a query Q' for which a dynamic program has already been obtained. Such a reduction needs to be consistent with first-order logic and its use in this dynamic context. A suitable kind of reductions are *bounded first-order reductions*. Intuitively, a query Q reduces to a query Q' via a bounded first-order reduction if a modification of an instance of Q induces constantly many, first-order definable modifications in an instance of Q' . Note that if Q can be reduced to Q' via a bounded first-order reduction, then first-order update formulas for a modification of an instance for Q can be obtained by composing the first-order update formulas for the corresponding (first-order definable) modifications of the instance of Q' . We refer to [16] and [12] for a detailed exposition to bounded first-order reductions.

In this article we study dynamic programs for queries on (labeled) graphs. For most of our dynamic programs the precise encoding of graphs is not important. If the input to a query is a single Σ -labeled graph $G = (V, E)$ then it can, for example, be encoded by binary relations E_σ that store all σ -labeled edges, for all $\sigma \in \Sigma$. Similarly for constantly many graphs. Some of our results are for input databases that contain more than constantly many graphs. Those can be encoded in higher arity relations in a straightforward way. For example, linearly many graphs can be stored in ternary relations E_σ containing a tuple (g, u, v) if graph g contains a σ -labeled edge (u, v) .

Graph databases and query languages We review basic definitions of graph databases in order to fix notations and introduce the query languages used in this work.

A *graph database* over an alphabet Σ is a finite Σ -labeled graph $G = (V, E)$ where V is a finite set of nodes and E is a set of labeled edges $(u, \sigma, v) \subseteq V \times \Sigma \times V$. Here σ is called the *label* of edge (u, σ, v) . Given a Σ -labeled graph $G = (V, E)$ and a symbol $\sigma \in \Sigma$, we denote by G_σ the projection of G onto its σ -labeled edges, that is, the graph G_σ has the edge set $\{(u, v) \mid (u, \sigma, v) \in E\}$. We say that a Σ -labeled graph G is *acyclic* if the graph $\cup_{\sigma \in \Sigma} G_\sigma$ is acyclic, and *undirected*, if for each $\sigma \in \Sigma$ the graph G_σ is undirected.

² In [25, 24, 22] the class DYNFO comes with an arbitrary initialization, yet there the focus is on lower bounds.

A *path* ρ in G from v_0 to v_m is a sequence of edges $(v_0, \sigma_1, v_1), \dots, (v_{m-1}, \sigma_m, v_m)$ of G , for some length $m \geq 0$. The *label* of ρ , denoted by $\lambda(\rho)$, is the word $\sigma_1 \dots \sigma_m \in \Sigma^*$. Paths of length zero are labeled by the empty string ϵ . For a formal language $L \subseteq \Sigma^*$, we say that ρ is an L -path if $\lambda(\rho) \in L$.

The basic building block of many graph query languages are *regular path queries* (short: RPQs). An RPQ selects all pairs of nodes in a Σ -labeled graph that are connected by an L -path, for a given regular language $L \subseteq \Sigma^*$. Here we are interested in two extensions of regular path queries. One of them are path queries defined by non-regular languages, namely context-free and non-context-free languages.

The second extension to be studied, *extended conjunctive regular path queries* (short: ECRPQs), allows to define multiple paths and to compare their labels based on relations on words. In the following we give a short introduction to ECRPQs and refer to [4] for a detailed study.

In ECRPQs, paths are compared by *regular relations*. A k -ary regular relation R over alphabet Σ is defined by a finite state automaton \mathcal{A} that synchronously reads k words over $\Sigma \cup \perp$, with $\perp \notin \Sigma$. The \perp symbol is a padding symbol that may only occur at the end of a word, and therefore allows for processing words of different length. More formally \mathcal{A} reads words over the alphabet $(\Sigma \cup \perp)^k$, and a k -tuple of words is in R if its corresponding string over $(\Sigma \cup \perp)^k$ is accepted by \mathcal{A} .

An ECRPQ is of the form $\mathcal{Q}(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} R_j(\vec{\omega}_j)$ where

- each R_j is a regular relation over Σ (specified by some finite state automaton),
- $\vec{x} = (x_1, \dots, x_m)$, $\vec{y} = (y_1, \dots, y_m)$ and \vec{z} are tuples of node variables such that the variables in \vec{z} occur in \vec{x} or \vec{y} , and
- $\vec{\pi} = (\pi_1, \dots, \pi_m)$ and $\vec{\omega}_1, \dots, \vec{\omega}_t$ are distinct tuples of path variables such that all variables in each $\vec{\omega}_j$ occur in $\vec{\pi}$.

In general, both node and path variables can occur in the head of an ECRPQ. Outputs of ECRPQs are potentially infinite sets then, since there can be infinitely many paths in graphs with cycles. Nevertheless, given a Σ -labeled graph $G = (V, E)$ and a tuple \vec{v} of nodes, the answer set is a regular relation over the alphabet $(V^k \cup \Sigma_{\perp}^k)$, where k is the number of path variables in the head. Such a regular relation is used as an encoding of all possible path outputs. For a fixed ECRPQ \mathcal{Q} , given G and \vec{v} , the automaton for this regular relation can be obtained in PTIME [4]. More precisely, it is definable by first-order queries evaluated on the graph database. We can therefore neglect path variables in the dynamic setting—for every tuple of nodes in the answer of the query we can obtain the regular relation encoding the output paths by first order queries.

The semantics of ECRPQs is defined in a natural way. For an ECRPQ \mathcal{Q} of the above form, a Σ -labeled graph $G = (V, E)$, and mappings ν from node variables to nodes and μ from path variables to paths, we write $(G, \nu, \mu) \models \mathcal{Q}$ if

- $\mu(\pi_i)$ is a path in G from $\nu(x_i)$ to $\nu(y_i)$ for $1 \leq i \leq m$, and
- the tuple $(\lambda(\mu(\pi_{j_1})), \dots, \lambda(\mu(\pi_{j_k})))$ belongs to the relation R_j for each $\vec{\omega}_j = (\pi_{j_1}, \dots, \pi_{j_k})$.

The result of \mathcal{Q} evaluated on G is defined by $Q(G) \stackrel{\text{def}}{=} \{\nu(\vec{z}) : (G, \nu, \mu) \models \mathcal{Q}\}$.

Product Graphs and Graph Query Languages There is a strong connection between the evaluation problem for many graph query languages and the reachability query for products of labeled graphs. For example, the evaluation of a regular path query L on a labeled graph G can be reduced to reachability in the product graph $\mathcal{A} \times G$ where \mathcal{A} is a finite state

automaton for L . Product graphs also help for the evaluation of fragments of ECRPQs as well. We will exploit this connection at several places and therefore present some basic properties of product graphs next.

The *product graph* $\prod_i G_i$ of m Σ -labeled graphs $G_i = (V_i, E_i)$, $1 \leq i \leq m$, has nodes $\prod_i V_i$ and an edge (\vec{x}, \vec{y}) between two nodes $\vec{x} = (x_1, \dots, x_m)$ and $\vec{y} = (y_1, \dots, y_m)$ if there is a symbol $\sigma \in \Sigma$ such that $(x_i, \sigma, y_i) \in E_i$ for each $1 \leq i \leq m$. The graphs G_i are called *factors* of the graph product. Graph products for unlabeled graphs are defined analogously. The following well known property characterizes reachability in (labeled) product graphs.

► **Fact 5.** *Let $(G_i)_{1 \leq i \leq m}$ be graphs (Σ -labeled graphs) with $G_i = (V_i, E_i)$ and let $\vec{x} = (x_1, \dots, x_m), \vec{y} = (y_1, \dots, y_m)$ be two pairs of nodes of $\prod_i G_i$. Then \vec{y} is reachable from \vec{x} in $\prod_i G_i$ if and only if there are paths ρ_i from x_i to y_i in G_i with $|\rho_i| \leq |\prod_i V_i|$, for $i \in \{1, \dots, m\}$, and $|\rho_i| = |\rho_j|$ ($\lambda(\rho_i) = \lambda(\rho_j)$) respectively for all $i, j \in \{1, \dots, m\}$. ◀*

The preceding fact can be used in the dynamic context as well, i.e. it is compatible with bounded first-order reductions. More precisely, reachability in products of unlabeled graphs can be inferred from all distances in the factors. We say that *all distances up to n^c* , for $c \in \mathbb{N}$, are computed by a dynamic program if, for a graph G with n nodes and arithmetic on the domain³, it maintains a relation D that contains all tuples (x, y, ℓ) such that there is a path from x to y of length ℓ , for $0 \leq \ell \leq n^c$.

► **Proposition 6.** *The following problems are equivalent under bounded first-order reductions with built-in arithmetic:*

- (a) *Maintaining all distances up to n^2 .*
- (b) *Maintaining reachability in the product of two graphs (both of them subject to modifications).*
- (c) *Maintaining reachability in the product of two graphs, one of them a fixed path.*

Proof sketch. Problem (c) is clearly a special case of Problem (b). The reduction from Problem (b) to Problem (a) is an immediate consequence of Fact 5: two nodes (x, x') and (y, y') of a product graph $G \times G'$ are connected if and only if there are equal-length paths of length at most n^2 from x to y in G and from x' to y' in G' .

Thus it remains to reduce Problem (a) to Problem (c). For a graph G over domain $D \stackrel{\text{def}}{=} \{0, \dots, n-1\}$, consider the product graph $G \times P$ where P is the path $\{(0, 1), \dots, (n^2-1, n^2)\}$ (as usual numbers larger than n are encoded as tuples over D). Then there is a path of length ℓ between two nodes x and y of G if and only if there is a path from $(x, 0)$ to (y, ℓ) in $G \times P$. Furthermore, the path P is never modified. ◀

A similar equivalence can be established for problems related to reachability in products of Σ -labeled graphs:

► **Proposition 7.** *The following problems are equivalent under bounded first-order reductions:*

- (a) *Maintaining the existence of equally labeled paths between two pairs of nodes.*
- (b) *Maintaining reachability in the product of two Σ -labeled graphs.*
- (c) *Maintaining reachability in the product of two Σ -labeled graphs, one of them undirected.*
- (d) *Maintaining the palindrome path query on Σ -labeled graphs.*

³ We note that from the arithmetic on the domain, arithmetic upto n^c can be defined using first-order formulas.

Proof sketch. The equivalence of problems (a) and (b) is an immediate consequence of Fact 5.

We next show the equivalence of Problems (b) and (c). Clearly, Problem (c) is a special case of (b). For reducing Problem (b) to Problem (c) consider two (directed) Σ -labeled graphs G_1 and G_2 . Let $\# \notin \Sigma$ be a fresh symbol, and denote $\Sigma_{\#} = \Sigma \cup \{\#\}$. In a first step, from G_1 and G_2 we construct two undirected $\Sigma_{\#}$ -labeled graphs G'_1 and G'_2 such that (1) there is a path between two nodes of $G_1 \times G_2$ if and only if there is a $(\Sigma \circ \{\#\})^*$ -labeled path between the corresponding nodes in $G'_1 \times G'_2$, and (2) one modification in G_i corresponds to at most two modifications in G'_i definable in first-order. To this end, the graph G'_i has two nodes x_{in} and x_{out} for each node x of G_i . An edge (x, σ, y) of G_i is encoded by the edges $(x_{\text{out}}, \sigma, y_{\text{in}})$ and $(y_{\text{in}}, \#, y_{\text{out}})$ in G'_i . (In particular, the edge $(y_{\text{in}}, \#, y_{\text{out}})$ is present in G'_i as soon as y has an incoming edge in G_i .)

Now every path in $G_1 \times G_2$ corresponds to a $(\Sigma \circ \{\#\})^*$ -labeled path in $G'_1 \times G'_2$, which in turn corresponds to a path in the product graph $G'_1 \times G'_2 \times \mathcal{A}$ where \mathcal{A} is the labeled (directed) graph $\bullet \xrightarrow[\#]{\Sigma} \bullet$ representing the language $(\Sigma \circ \{\#\})^*$. Since $G'_1 \times G'_2 \times \mathcal{A}$ is the product of the undirected graph G'_1 and the directed graph $G'_2 \times \mathcal{A}$, this yields the intended reduction (as one modification in G_2 yields at most six modifications in $G'_2 \times \mathcal{A}$).

We now show that the problems (b) and (d) are equivalent. For reducing (d) to (b) we consider, for simplicity, only palindromes of even length; the construction can be easily adapted to arbitrary palindromes. Let G be a labeled (directed) graph. Then there is a path from x to y labeled by a palindrome ww^R if and only if there is a node z such that there are w and w^R labeled paths from x to z and from z to y , respectively. Thus finding a palindromic path from x to y corresponds to finding a node z such that there is a path from (x, y) to (z, z) in the product graph $G \times G^-$, where G^- denotes the graph obtained from G by reversing each of its edges (definable in first-order). Note that one modification of G corresponds to two modifications in the factors of $G \times G^-$.

For the other direction, let G_1 and G_2 be two arbitrary Σ -labeled graphs and let $\# \notin \Sigma$ be a fresh symbol. We assume, without loss of generality, that the node sets of the graphs are disjoint. There is a path from (x_1, x_2) to (y_1, y_2) in $G_1 \times G_2$ if and only if there is a word w , a w -labeled path from x_1 to y_1 and a w -labeled path from x_2 to y_2 . The latter condition is equivalent to the existence of a $w\#w^R$ -labeled path in the graph $G \stackrel{\text{def}}{=} G_1 \cup G_2^-$ extended by the edge $(y_1, \#, y_2)$. \blacktriangleleft

3 Dynamic Path Queries

Path queries, as mentioned in the introduction, have almost not been studied in dynamic complexity before. Until recently not even the simple query induced by the language $L(a^*)$ was known to be in DYNFO. Yet as an immediate consequence of the dynamic first-order update program for reachability exhibited in [6], all fixed regular path queries (and, since DYNFO is closed under conjunctions, also conjunctions of them) can be maintained by first-order update formulas.

In this section we continue the exploration of the dynamic maintainability of path queries. We show that under insertions quantifier-free update formulas are sufficient to maintain (fixed) regular path queries, and that more expressive path queries can be maintained for restricted classes of graphs and constrained modifications.

► **Theorem 8.** *When only insertions are allowed then every regular path query can be maintained by quantifier-free update formulas.*

We conjecture that quantifier-free update formulas do not suffice to maintain RPQs under both insertions and deletions. This would imply that reachability can be maintained without quantifiers which seems to be very unlikely. A first step towards verifying this conjecture was done in [25] where it was shown that reachability cannot be maintained with binary quantifier-free programs.

Proof. The following notion will be useful. Let \mathcal{A} be a deterministic finite state automaton (short: DFA) and let G be a labeled graph. Then a path ρ in G can be *read by \mathcal{A} starting in a state p and ending in a state q* if \mathcal{A} can reach state q from state p by reading the label sequence $\lambda(\rho)$ of ρ .

Let L be a regular path query and let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be a DFA with $L = L(\mathcal{A})$. We construct a DYNPROP-program \mathcal{P} that maintains L .

The program \mathcal{P} has input schema $\{E_\sigma \mid \sigma \in \Sigma\}$ and an auxiliary schema that contains a binary relation symbol $R_{p,q}$ for every pair $(p, q) \in Q^2$ of states, as well as a binary designated query symbol R . The simple idea is that in a state \mathcal{S} with underlying labeled graph G , the relation $R_{p,q}^{\mathcal{S}}$ contains all tuples $(x, y) \in V^2$ such that \mathcal{A} , for some labeled path ρ from x to y , can read ρ by starting in state p and ending in state q .

The update formulas for the relations $R_{p,q}$ are slightly more involved than the formulas for maintaining reachability under insertions. This is because \mathcal{A} might reach a state q from a state p only by reading a labeled path from x to y that contains one or more loops. The crucial observation is, however, that for deciding whether (x, y) is in $R_{p,q}$ it suffices to consider paths that contain the node x at most $|Q|$ times (as paths that contain x more than $|Q|$ times can be shortened). This suffices to maintain the relations $R_{p,q}$ dynamically.

The update formulas for $R_{p,q}$ and R are as follows:

$$\begin{aligned} \phi_{\text{INS}_{E_\sigma}}^{R_{p,q}}(u, v; x, y) &\stackrel{\text{def}}{=} R_{p,q}(u, v) \vee \bigvee_{p', q'} \left(R_{p,p'}(x, u) \wedge \varphi_{p', q'}^{|Q|}(u, v) \wedge R_{q', q}(v, y) \right) \\ \phi_{\text{INS}_{E_\sigma}}^R(u, v; x, y) &\stackrel{\text{def}}{=} \bigvee_{f \in F} \phi_{\text{INS}_{E_\sigma}}^{R_{s, f}}(u, v; x, y) \end{aligned}$$

Here the formula $\varphi_{p', q'}^{|Q|}(u, v)$ shall only be satisfied by tuples (u, v) for which there exists a path ρ from u to v such that \mathcal{A} can read ρ by starting in p' and ending in q' . It shall be satisfied by all such tuples with a witness path ρ that contains node u at most $|Q|$ times.

We inductively define, for every $1 \leq i \leq |Q|$ and all $p, q \in Q$, the slightly more general formulas $\varphi_{p, q}^i(u, v)$ as follows:

$$\begin{aligned} \varphi_{p, q}^1(u, v) &\stackrel{\text{def}}{=} [(p, \sigma, q) \in \delta] \vee R_{p, q}(u, v) \\ \varphi_{p, q}^i(u, v) &\stackrel{\text{def}}{=} \varphi_{p, q}^{i-1}(u, v) \vee \bigvee_{p', q'} \left(\varphi_{p, p'}^1(u, v) \wedge R_{p', q'}(v, u) \wedge \varphi_{q', q}^{i-1}(u, v) \right) \end{aligned}$$

◀

Capturing non-regular path queries by first-order update formulas seems to be significantly harder than capturing CRPQs. We provide only some preliminary results for restricted classes of graphs and modifications.

When all distances for all pairs of nodes can be maintained for a restricted class of graphs, then also non-regular and even non-contextfree path queries can be maintained (e.g. the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$).

► **Theorem 9.** (a) *There is a non-context-free path query that can be maintained in DYNFO on acyclic and undirected Σ -labeled graphs.*

- (b) *There is a non-context-free path query that can be maintained in DYNFO when only insertions are allowed.*

Proof. The non-context-free path query induced by $L = \{a^n b^n c^n\}$ can be maintained since for a graph G distances on G_a, G_b, G_c can be kept up-to-date for those restrictions (see Theorem 12 and Theorem 13). The arithmetic needed for those theorems can be simulated by Proposition 4. We note that also path queries induced by languages such as $\{a^n b^{n+m} c^m\}$ can be maintained by first-order update formulas. \blacktriangleleft

On acyclic graphs, all context-free path queries can be maintained. It is known that context-free languages are in DYNFO [11] and that the Dyck language with two types of parentheses can be maintained on acyclic graphs [20]. Generalizing the techniques used for those two results yields the following theorem.

► **Theorem 10.** *All context-free path queries can be maintained in DYNFO on acyclic graphs.*

To prove Theorem 10, we fix a context-free language L and a grammar $\mathcal{G} = (V, \Sigma, S, P)$ for L . We assume, without loss of generality, that \mathcal{G} is in Chomsky normal form, that is, it has only rules of the form $X \rightarrow YZ$ and $X \rightarrow \sigma$. Furthermore, if $\epsilon \in L$ then $S \rightarrow \epsilon \in P$ and no right-hand side of a rule contains S . We write $Z \Rightarrow^* w$ if $w \in (\Sigma \cup V)^*$ can be derived from $Z \in V$ using rules of \mathcal{G} .

The dynamic program maintaining L on acyclic graphs will use 4-ary auxiliary relation symbols $R_{Z \rightarrow Z'}$ for all $Z, Z' \in V$. The intention is that in every state \mathcal{S} with input database G , the relation $R_{Z \rightarrow Z'}^{\mathcal{S}}$ contains a tuple (x_1, y_1, x_2, y_2) if and only if there are strings $s_1, s_2 \in \Sigma^*$ such that $Z \Rightarrow^* s_1 Z' s_2$ and there is an s_i -path ρ_i from x_i to y_i for $i \in \{1, 2\}$. The paths ρ_1 and ρ_2 are called *witnesses* for $(x_1, y_1, x_2, y_2) \in R_{Z \rightarrow Z'}^{\mathcal{S}}$. Later we will see that whether two nodes are connected by an L -path after an update can be easily verified using those relations.

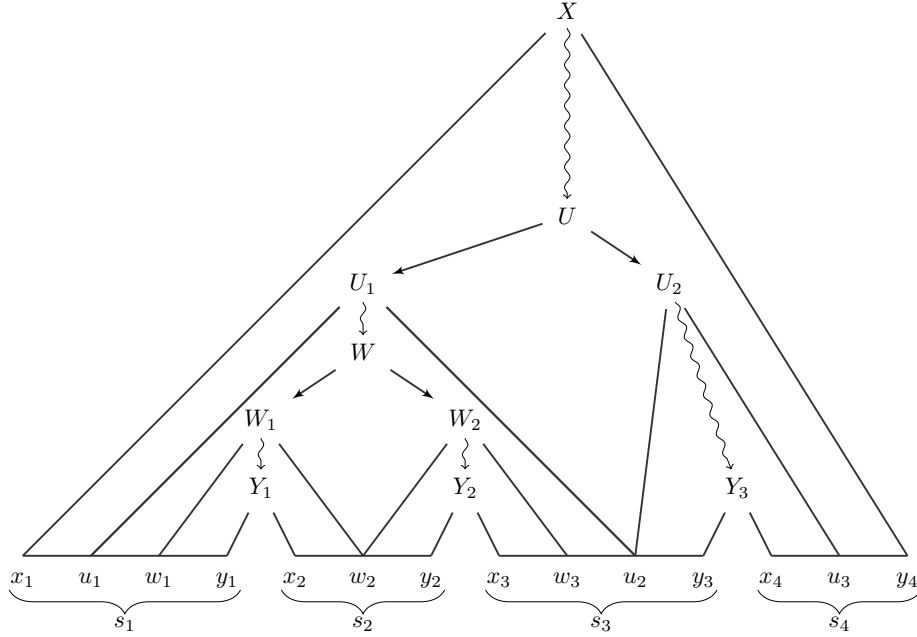
It turns out that for updating the relations $R_{Z \rightarrow Z'}^{\mathcal{S}}$, it is necessary to have access to $(2k+2)$ -ary relations $R_{X \rightarrow Y_1, \dots, Y_k}^{\mathcal{S}}$, for $k \in \{1, 2, 3\}$, which contain a tuple $(x_1, y_1, \dots, x_{k+1}, y_{k+1})$ if and only if there are strings $s_1, \dots, s_{k+1} \in \Sigma^*$ such that $X \Rightarrow^* s_1 Y_1 s_2 \dots s_k Y_k s_{k+1}$ and there is an s_i -path ρ_i from x_i to y_i in the input database underlying \mathcal{S} .

Next, in Lemma 11, we prove that every relation $R_{X \rightarrow Y_1, \dots, Y_k}^{\mathcal{S}}$ is first-order definable from the relations $R_{Z \rightarrow Z'}^{\mathcal{S}}$ (and thus only relations $R_{Z \rightarrow Z'}^{\mathcal{S}}$ have to be stored as auxiliary data). This lemma is inspired by Lemma 7.3 from [20], and its proof is a generalization of the technique used in the proof of Theorem 4.1 in [11]. Afterwards we prove Theorem 10 by showing how to use the relations $R_{Z \rightarrow Z'}^{\mathcal{S}}$ to maintain L and how to update the relations $R_{Z \rightarrow Z'}^{\mathcal{S}}$ using the formulas that define relations of the form $R_{X \rightarrow Y_1, Y_2}^{\mathcal{S}}$ and $R_{X \rightarrow Y_1, Y_2, Y_3}^{\mathcal{S}}$.

► **Lemma 11.** *For a grammar \mathcal{G} in Chomsky normal form, $k \geq 2$ and variables X, Y_1, \dots, Y_k there is a first-order formula $\varphi_{X \rightarrow Y_1, \dots, Y_k}$ over schema $\tau = \{R_{Z \rightarrow Z'} \mid Z, Z' \in V\}$ that defines $R_{X \rightarrow Y_1, \dots, Y_k}^{\mathcal{S}}$ in states \mathcal{S} where the relations $R_{Z \rightarrow Z'}^{\mathcal{S}}$ are as described above.*

Proof sketch. We explain how $\varphi_{X \rightarrow Y_1, Y_2, Y_3}$ tests whether a tuple is contained in $R_{X \rightarrow Y_1, Y_2, Y_3}^{\mathcal{S}}$. The construction for general k is analogous.

If a tuple $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ is contained in $R_{X \rightarrow Y_1, Y_2, Y_3}^{\mathcal{S}}$ witnessed by s_i -paths ρ_i from x_i to y_i such that $X \Rightarrow^* s_1 Y_1 s_2 Y_2 s_3 Y_3 s_4$, then in the derivation tree of $s_1 Y_1 s_2 Y_2 s_3 Y_3 s_4$ from X there is a variable U such that $U \rightarrow U_1 U_2$ and either (1) Y_1 and Y_2 are derived from U_1 , and Y_3 is derived from U_2 ; or (2) Y_1 is derived from U_1 , and Y_2 and Y_3 are derived from U_2 . In case (1), the derivation subtree starting from U_1 contains a variable W such that $W \rightarrow W_1 W_2$ and Y_1 is derived from W_1 and Y_2 is derived from W_2 . Analogously for case (2). The derivation tree of X for case (1) is illustrated in Figure 1.



■ **Figure 1** Illustration of when a tuple $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ is contained in $R_{X \rightarrow Y_1, Y_2, Y_3}$ in Lemma 11.

The formula $\varphi_{X \rightarrow Y_1, Y_2, Y_3}$ is the disjunction of formulas ψ_1 and ψ_2 , responsible for dealing with the cases (1) and (2) respectively. We only exhibit ψ_1 , the formula ψ_2 can be constructed analogously. The formula ψ_1 guesses the variables U, U_1, U_2, W, W_1 and W_2 , and the start and end positions of strings derived from those variables. Whether $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ is contained in $R_{X \rightarrow Y_1, Y_2, Y_3}^S$ can then be tested using the relations $R_{Z \rightarrow Z'}$. For simplicity the formula ψ_1 reuses the element names x_i and y_i as variable names and is defined as follows:

$$\begin{aligned} \psi_1(x_1, y_1, \dots, x_4, y_4) = & \exists u_1 \exists u_2 \exists u_3 \bigvee_{\substack{U, U_1, U_2 \in V \\ U \rightarrow U_1 U_2 \in P}} \exists w_1 \exists w_2 \exists w_3 \bigvee_{\substack{W, W_1, W_2 \in V \\ W \rightarrow W_1 W_2 \in P}} \\ & \left(R_{X \rightarrow U}(x_1, u_1, u_3, y_4) \wedge R_{U_1 \rightarrow W}(u_1, w_1, w_3, u_2) \right. \\ & \wedge R_{W_1 \rightarrow Y_1}(w_1, y_1, x_2, w_2) \wedge R_{W_2 \rightarrow Y_2}(w_2, y_2, x_3, w_3) \\ & \left. \wedge R_{U_2 \rightarrow Y_3}(u_2, y_3, x_4, u_3) \right) \end{aligned}$$

◀

We now use the relations $R_{Z \rightarrow Z'}$ and the formulas $\varphi_{X \rightarrow Y_1, Y_2, Y_3}$ for maintaining context-free path queries on acyclic graphs.

Proof idea (of Theorem 10). Let L be an arbitrary context-free language and let $\mathcal{G} = (V, \Sigma, S, P)$ be a grammar for L in Chomsky normal form. We provide a DYNFO-program \mathcal{P} with designated binary query symbol Q that maintains L on acyclic graphs. The input schema is $\{E_\sigma \mid \sigma \in \Sigma\}$ and the auxiliary schema is $\tau_{\text{aux}} = \{R_{X \rightarrow Y} \mid X, Y \in V\} \cup \{T\}$. The intention of the auxiliary relation symbols $R_{X \rightarrow Y}$ has already been explained above; the relation symbol T shall store the transitive closure of the input graph (where the input graph is the union of all E_σ).

Before showing how to update the relations $R_{X \rightarrow Y}$, we state the update formulas for the query relation Q . The update formulas distinguish whether the witness path is of length 0 or of length at least 1. The updated relations $R_{X \rightarrow Y}$ are used for the latter case.

$$\begin{aligned}
\phi_{\text{INS}_{E_\sigma}}^Q(u, v; x, y) &\stackrel{\text{def}}{=} ([S \rightarrow \epsilon \in P] \wedge x = y) \\
&\quad \vee \exists z_1 \exists z_2 \bigvee_{\substack{U \in V \\ U \rightarrow \tau \in P}} (\phi_{\text{INS}_{E_\sigma}}^{R_{S \rightarrow U}}(u, v; x, z_1, z_2, y) \wedge E_\tau(z_1, z_2)) \\
&\quad \vee \bigvee_{\substack{U \in V \\ U \rightarrow \sigma \in P}} (\phi_{\text{INS}_{E_\sigma}}^{R_{S \rightarrow U}}(u, v; x, u, v, y)) \\
\phi_{\text{DEL}_{E_\sigma}}^Q(u, v; x, y) &\stackrel{\text{def}}{=} ([S \rightarrow \epsilon \in P] \wedge x = y) \\
&\quad \vee \exists z_1 \exists z_2 \bigvee_{\substack{U \in V \\ \tau \neq \sigma \\ U \rightarrow \tau \in P}} (\phi_{\text{DEL}_{E_\sigma}}^{R_{S \rightarrow U}}(u, v; x, z_1, z_2, y) \wedge E_\tau(z_1, z_2)) \\
&\quad \vee \exists z_1 \exists z_2 \bigvee_{\substack{U \in V \\ U \rightarrow \sigma \in P}} (\phi_{\text{DEL}_{E_\sigma}}^{R_{S \rightarrow U}}(u, v; x, z_1, z_2, y) \wedge E_\sigma(z_1, z_2) \wedge (z_1 \neq u \vee z_2 \neq v))
\end{aligned}$$

It remains to present update formulas for each $R_{X \rightarrow Y}$. For simplicity we identify names of variable and elements.

After inserting a σ -edge (u, v) , a tuple (x_1, y_1, x_2, y_2) is contained in $R_{X \rightarrow Y}$ if there are two witness paths ρ_1 and ρ_2 such that (1) ρ_1 and ρ_2 have already been witnesses before the insertion, or (2) only ρ_1 uses the new σ -edge, or (3) only ρ_2 uses the new σ -edge, or (4) both ρ_1 and ρ_2 use the new σ -edge. In case (2) the path ρ_1 can be split into a path from x_1 to u , the edge (u, v) and a path from v to y_1 . Similarly in the other cases and for ρ_2 . Using the formulas from Lemma 11 this can be expressed as follows:

$$\phi_{\text{INS}_{E_\sigma}}^{R_{X \rightarrow Y}}(u, v; x_1, y_1, x_2, y_2) \stackrel{\text{def}}{=} R_{X \rightarrow Y}(x_1, y_1, x_2, y_2) \vee \quad (1)$$

$$\bigvee_{\substack{U_1, U_2 \in V \\ U_1 \rightarrow \sigma \in P \\ U_2 \rightarrow \sigma \in P}} (\varphi_{X \rightarrow U_1, Y}(x_1, u, v, y_1, x_2, y_2)) \quad (2)$$

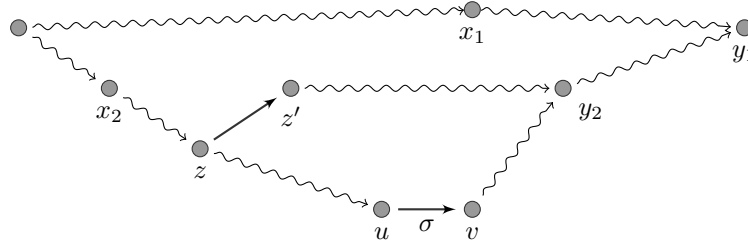
$$\vee \varphi_{X \rightarrow Y, U_2}(x_1, y_1, x_2, u, v, y_2) \quad (3)$$

$$\vee \varphi_{X \rightarrow U_1, Y, U_2}(x_1, u, v, y_1, x_2, u, v, y_2) \quad (4)$$

After deleting a σ -edge (u, v) a tuple (x_1, y_1, x_2, y_2) is in $R_{X \rightarrow Y}$ if it still has witness paths ρ_1 and ρ_2 from x_1 to y_1 and from x_2 to y_2 , respectively. The update formula for $R_{X \rightarrow Y}$ verifies that such witness paths exist. Therefore, similar to Example 3, the formula distinguishes for each $i \in \{1, 2\}$ whether (1) there was no path from x_i to y_i via (u, v) before deleting the σ -edge (u, v) , or (2) there was a path from x_i to y_i via (u, v) . See Figure 2 for an illustration.

In case (1) all paths present from x_i to y_i before the deletion of the σ -edge (u, v) are also present after the deletion. In particular the set of possible witnesses ρ_i remains the same. For case (2), the update formula has to check that there is still a witness path ρ_i . Such a path ρ_i has the options (a) to still use the edge (u, v) but for a $\tau \neq \sigma$, and (b) to not use the edge (u, v) at all.

The update formula for $R_{X \rightarrow Y}$ is a disjunction over all those cases for the witnesses for (x_1, y_1) and (x_2, y_2) . Instead of presenting formulas for all those cases, we explain the idea for two representative cases. All other cases are analogous.



■ **Figure 2** Illustration of the update of $R_{X \rightarrow Y}$ after deletion of σ -edge (u, v) in the proof of Lemma 11. The nodes x_1 and y_1 satisfy Condition (1), whereas nodes x_2 and y_2 satisfy Condition (2).

We first look at the case where (x_1, y_1) satisfies (1), (x_2, y_2) satisfies (2) and there are witness paths ρ_1 and ρ_2 where ρ_2 satisfies (a). The following formula deals with this case:

$$\begin{aligned}
 & (\neg T(x_1, u) \vee \neg T(v, y_1)) \wedge T(x_2, u) \wedge T(v, y_2) \\
 & \wedge \bigvee_{\substack{\tau \neq \sigma, U_2 \in V \\ U_2 \rightarrow \tau \in P}} (\varphi_{X \rightarrow Y, U_2}(x_1, y_1, x_2, u, v, y_2) \wedge E_\tau(u, v))
 \end{aligned}$$

In the first line the premises for this case are checked, in the second line it is verified that ρ_2 uses τ -edge (u, v) for $\sigma \neq \tau$.

Now we consider the case where both (x_1, y_1) as well as (x_2, y_2) satisfy (2), and where there are witness paths ρ_1 and ρ_2 where ρ_1 satisfies (a) and ρ_2 satisfies (b). The existence of such a path ρ_1 can be verified as above. For verifying the existence of such a path ρ_2 , a path not using (u, v) has to be found. This is achieved by relying on the same technique as for maintaining reachability for acyclic graphs (see Example 3). The following formula verifies the existence of such ρ_1 and ρ_2 :

$$\begin{aligned}
 & T(x_1, u) \wedge T(v, y_1) \wedge T(x_2, u) \wedge T(v, y_2) \\
 & \wedge \exists z \exists z' \bigvee_{\substack{\tau \neq \sigma, U_1, U_2 \in V \\ U_1 \rightarrow \tau \in P \\ U_2 \rightarrow \tau' \in P}} \left(\varphi_{X \rightarrow U_1, Y, U_2}(x_1, u, v, y_1, x_2, z, z', y_2) \wedge E_\tau(u, v) \right. \\
 & \quad \wedge (T(x_2, z) \wedge E_{\tau'}(z, z') \wedge (z \neq u \vee z' \neq v) \\
 & \quad \left. \wedge T(z', y_2) \wedge T(z, u) \wedge \neg T(z', u) \right)
 \end{aligned}$$

Again, in the first line the premises for this case are checked. In the second line z and z' are chosen with the purpose to find an alternative path ρ_2 (as in Example 3), and it is verified that ρ_1 and ρ_2 are witness paths. The third and forth lines verify that z and z' yield an alternative path. ◀

4 Dynamic Extended Conjunctive Regular Path Queries

In this section we explore the maintainability of ECRPQs. In contrast to path queries, ECRPQs allow for testing properties of tuples of paths between pairs of nodes. Comparing the length of two paths is one of the simplest such properties and is therefore studied first. Afterwards we extend some of the techniques developed for maintaining the lengths of paths to ECRPQs.

4.1 Maintaining Distances

Maintaining all distances in arbitrary graphs is one of the big challenges of dynamic complexity. Recall that for maintaining all distances up to n^c a dynamic program has to update, for a graph G , a relation D that contains all tuples (x, y, ℓ) such that there is a path from x to y of length ℓ in G , for $0 \leq \ell \leq n^c$.

The recent dynamic algorithm for maintaining reachability (see [6]) does, unfortunately, not offer hints at how to maintain distances. A dynamic upper bound for distances is provided by Hesse's DYN TC^0 -program for reachability [13]. The program actually maintains the number of different paths of length ℓ between every pair of nodes, for any length ℓ up to the size of the graph, and thus all distances for all pairs of nodes. The program can be easily modified to compute all distances up to fixed polynomials.

Here we present preliminary results for maintaining all distances with first-order formulas for restricted modifications as well as for restricted classes of graphs. Furthermore we show that distances cannot be maintained with quantifier-free update formulas.

The *shortest* distance between every pair of nodes can be easily maintained in DYNFO when edges can only be inserted; basically because shortest paths do not contain loops. Maintaining *all* distances for all pairs of nodes under insertions requires some work.

► **Theorem 12.** *All distances up to $p(n)$ can be maintained in DYNFO under insertions for every fixed polynomial $p(n)$.*

Proof. We describe how to maintain distances up to n ; the generalization to distances up to $p(n)$ is straightforward and sketched at the end of the proof. The idea is to maintain a 4-ary relation A that contains a tuple (x, y, t, ℓ) if there are t (not necessarily distinct) paths from x to y such that the sum of their lengths is ℓ .

There is a path of length ℓ from node x to node y if and only if $(x, y, 1, \ell)$ holds. For maintaining this information, we need the full relation: a path from x to y can use a newly inserted edge (u, v) several times if cycles are present. Also, the path can use an arbitrary combination of cycles including that edge, and each cycle can be used arbitrarily often.

When inserting an edge (u, v) the updated relation A is defined by the following formula:

$$\begin{aligned} \phi_{\text{INS}_E}^A(u, v; x, y, t, \ell) \stackrel{\text{def}}{=} & \exists t_- \exists t_+ \exists t_\circ \exists \ell_- \exists \ell_{+1} \exists \ell_{+2} \exists \ell_\circ \\ & \left(A(x, y, t_-, \ell_-) \wedge A(x, u, t_+, \ell_{+1}) \wedge A(v, y, t_+, \ell_{+2}) \wedge A(v, u, t_\circ, \ell_\circ) \right. \\ & \left. \wedge (t_+ = 0 \rightarrow t_\circ = 0) \wedge t_- + t_+ = t \wedge \ell_- + \ell_{+1} + \ell_{+2} + \ell_\circ + t_+ + t_\circ = \ell \right) \end{aligned}$$

If there are t paths with total length ℓ from x to y after the edge (u, v) is inserted, these paths can be divided into t_- paths that do not use the new edge (u, v) , with a total length of ℓ_- , and t_+ paths that use the edge (u, v) . Each one of these t_+ paths is composed of (i) one path from x to u that does not use (u, v) , (ii) the edge (u, v) , (iii) possibly some cycles from v back to v created by combining an old path from v to u and the new edge (u, v) , and (iv) one path from v to y that does not use (u, v) .

Without considering the cycles in v that use (u, v) , in total there are t_+ paths from x to u (with total length ℓ_{+1}), t_+ paths from v to y (with total length ℓ_{+2}) and t_+ times the new edge (u, v) . So these paths have total length $\ell_{+1} + \ell_{+2} + t_+$. Additionally, let t_\circ be the number of times the edge (u, v) is used in cycles from v to v in all t_+ paths together. These cycles can be obtained from t_\circ paths from v to u of total length ℓ_\circ and t_\circ times the new edge (u, v) . So in total, the t_+ paths have a total length of $\ell_{+1} + \ell_{+2} + t_+ + \ell_\circ + t_\circ$.

For maintaining distances upto $p(n)$, numbers of this magnitude are encoded by tuples of elements. Arithmetic upto $p(n)$ can be easily defined in a first-order fashion from the built-in arithmetic upto n . The above construction then translates in a straightforward way. ◀

Next we show that all distances for all pairs of nodes in undirected and acyclic graphs can be updated using first-order update formulas. For undirected graphs this slightly extends a result by Grädel and Siebertz [12] that the shortest distance can be maintained for undirected paths. For acyclic graphs the maintenance of all distances is a straight-forward extension of the dynamic program for maintaining reachability shown in Example 3.

► **Theorem 13.** *All distances up to $p(n)$ can be maintained in DYNFO for every fixed polynomial $p(n)$ for (a) undirected graphs, and (b) acyclic graphs.*

Proof. Again we describe how to maintain distances upto n only; the generalization to distances upto $p(n)$ is straightforward.

(a) We use the simple observation that if two nodes in an undirected graph G are connected by a path of length $m > 0$, then they are connected by a path of length $m + 2$ as well, since any edge of the path can be traversed repeatedly. A consequence is that all possible distances between two nodes x and y in an undirected graph can be easily determined if the shortest lengths d_o and d_e of paths of odd and even length between x and y are known: there is a path of length m if m is odd and $m \geq d_o$ or if m is even and $m \geq d_e$ (and if $x = y$, then x is no isolated node). Thus in order to maintain whether two nodes x and y are connected by a path of length m , it suffices (1) to maintain d_o and d_e and (2) to know whether m is even or odd.

The second part is easy since arithmetic is available. Maintaining d_o and d_e can be done by maintaining the shortest distances of pairs of nodes in the graph $G \times K_2$, where K_2 is the complete graphs on nodes $\{1, 2\}$. The shortest distance between $(u, 1)$ and $(v, 1)$ in $G \times K_2$ equals the length of the shortest even path from u to v in G , whereas the distance between $(u, 1)$ and $(v, 2)$ is equal to the length of the shortest odd one. Observe that an edge modification in G spans only two modifications in $G \times K_2$. Since shortest distances in an undirected graphs can be maintained in DYNFO [12], the result follows.

(b) This is a simple adaption of the maintenance procedure for the transitive closure of acyclic graphs (see Example 3). In addition to the transitive closure relation T , the dynamic program for distances in acyclic graphs maintains a ternary relation D that contains a tuple (x, y, ℓ) if and only if there is a path from x to y of length ℓ . The update formulas from Example 3 can be adapted easily by using the built-in arithmetic.

$$\begin{aligned} \phi_{\text{INS}_E}^D(u, v; x, y, \ell) &\stackrel{\text{def}}{=} D(x, y, \ell) \vee \exists d \exists d' (d + d' + 1 = \ell \\ &\quad \wedge D(x, u, d) \wedge D(v, y, d')) \\ \phi_{\text{DEL}_E}^D(u, v; x, y, \ell) &\stackrel{\text{def}}{=} T(x, y) \wedge \left(((\neg T(x, u) \vee \neg T(v, y)) \wedge D(x, y, \ell)) \right. \\ &\quad \vee \exists z \exists z' \exists d \exists d' (d + d' + 1 = \ell \wedge D(x, z, d) \wedge E(z, z') \\ &\quad \wedge (z \neq u \vee z' \neq v) \wedge D(z', y, d')) \\ &\quad \left. \wedge T(z, u) \wedge \neg T(z', u) \right) \end{aligned}$$

◀

In the rest of this subsection we discuss why distance information cannot be maintained by quantifier-free update formulas. So far the goal, when maintaining distances, was to store

tuples (a, b, ℓ) in some relation if there is a path from a to b of length ℓ , where the length ℓ referred to the built-in arithmetic. It can be easily seen that maintaining distances in this fashion is not possible with quantifier-free formulas (basically because a quantifier-free formula only has access to the numbers represented by the modified nodes).

Another way of maintaining distance information is to store a 4-relation that contains a tuple (a_1, a_2, b_1, b_2) if and only if there are paths from a_1 to a_2 and from b_1 to b_2 of equal length. We show that this relation cannot be maintained by quantifier-free programs.

Denote by EQUAL-LENGTH-PATHS the query on (unlabeled) graphs that selects all tuples (a_1, a_2, b_1, b_2) such that there are paths from a_1 to a_2 and from b_1 to b_2 of equal length.

► **Theorem 14.** *The query EQUAL-LENGTH-PATHS cannot be maintained by quantifier-free update formulas, even when the auxiliary relations can be initialized arbitrarily. In particular, ECRPQs and reachability in product graphs cannot be maintained in this setting either.*

Intuitively this is not very surprising. It is well known that non-regular languages and therefore, in particular, the language $\{a^n b^n \mid n \in \mathbb{N}\}$ cannot be maintained by a quantifier-free program [11]. Thus maintaining whether two isolated paths have the same length should not be possible either. Technical issues arise from the fact that the query EQUAL-LENGTH-PATHS is over graphs, not strings. Yet the techniques used for proving lower bounds for languages can be adapted.

We employ the following Substructure Lemma from [25, Lemma 4.1] which is a slight variation of Lemma 1 from [11].

The intuition of the Substructure Lemma is as follows. When updating an auxiliary tuple \vec{c} after an insertion or deletion of a tuple \vec{d} , a quantifier-free update formula has access to \vec{c} , \vec{d} , and the constants only. Thus if a sequence of modifications changes only tuples from a substructure \mathcal{A} of \mathcal{S} , then the auxiliary data of \mathcal{A} is not affected by information outside \mathcal{A} . In particular, two isomorphic substructures \mathcal{A} and \mathcal{B} remain isomorphic, when corresponding modifications are applied to them.

The notion of corresponding modifications is formalized as follows. Let π be an isomorphism from a structure \mathcal{A} to a structure \mathcal{B} . Two modifications $\delta(\vec{a})$ on \mathcal{A} and $\delta'(\vec{b})$ on \mathcal{B} are said to be π -respecting if $\delta = \delta'$ and $\vec{b} = \pi(\vec{a})$. Two sequences $\alpha = \delta_1 \cdots \delta_m$ and $\beta = \delta'_1 \cdots \delta'_m$ of modifications respect π if δ_i and δ'_i are π -respecting for every $i \leq m$. Recall that $P_\alpha(\mathcal{S})$ denotes the state obtained by executing the dynamic program \mathcal{P} for the modification sequence α from state \mathcal{S} .

► **Lemma 15** (Substructure Lemma [11]). *Let \mathcal{P} be a DYNPROP-program and let \mathcal{S} and \mathcal{T} be states of \mathcal{P} with domains S and T . Further let $A \subseteq S$ and $B \subseteq T$ such that $\mathcal{S} \upharpoonright A$ and $\mathcal{T} \upharpoonright B$ are isomorphic via π . Then $P_\alpha(\mathcal{S}) \upharpoonright A$ and $P_\beta(\mathcal{T}) \upharpoonright B$ are isomorphic via π for all π -respecting modification sequences α, β on A and B .*

Proof (of Theorem 14). Towards a contradiction, assume that $\mathcal{P} = (P, \text{INIT}, Q)$ is a dynamic program over schema $\tau = (\tau_{\text{in}}, \tau_{\text{aux}})$ that maintains the query EQUAL-LENGTH-PATHS in its designated query relation Q . Let n' be sufficiently large with respect to τ and n be sufficiently large with respect to n' . Further let m be the highest arity of a relation symbol from τ_{aux} .

Let $G = (V, E)$ be the empty graph with $|V| = n$ and let $\mathcal{S} = (V, E, \mathcal{A})$ be the state obtained by applying the initialization mapping of \mathcal{P} to G .

By Ramsey's Theorem for structures (see, e.g., [25, Theorem 4.3]) and because $n = |V|$ is sufficiently large with respect to n' there is a set $V' \subseteq V$ of size $2n'$ and an order \prec on V' such that all \prec -ordered m -tuples over V' are of equal atomic τ_{aux} -type. Let

us assume that $V' = A \cup B$ with $A = \{a_1, \dots, a_{n'}\}$ and $B = \{b_1, \dots, b_{n'}\}$, and that $a_1 \prec \dots \prec a_{n'} \prec b_1 \prec \dots \prec b_{n'}$.

Let $\mathcal{S}' \stackrel{\text{def}}{=} (V, E', \mathcal{A}')$ be the state of \mathcal{P} that is reached from \mathcal{S} after inserting the edges $(a_1, a_2), (a_2, a_3), \dots, (a_{n'-1}, a_{n'})$.

Our goal is to find i_1, i_2, i_3 with $i_1 < i_2 < i_3$ such that the substructures $\mathcal{S}' \upharpoonright \{a_{i_1}, a_{i_2}, b_1, \dots, b_{n'}\}$ and $\mathcal{S}' \upharpoonright \{a_{i_1}, a_{i_3}, b_1, \dots, b_{n'}\}$ are isomorphic. Then, in the state \mathcal{S}'' obtained from \mathcal{S}' by inserting the edges $\{(b_1, b_2), (b_2, b_3), \dots, (b_{i_2-i_1-1}, b_{i_2-i_1})\}$, the tuples $(a_{i_1}, a_{i_2}, b_1, b_{i_2-i_1})$ and $(a_{i_1}, a_{i_3}, b_1, b_{i_2-i_1})$ will either be both in Q or both not in Q (due to the Substructure Lemma). However, there is a path of length $i_2 - i_1$ between a_{i_1} and a_{i_2} but not from a_{i_1} to a_{i_3} , a contradiction.

It remains to exhibit such i_1, i_2 and i_3 . To this end observe that for all m -ary tuples \vec{b}_1 and \vec{b}_2 , the tuples (a_i, a_j, \vec{b}_1) and (a_i, a_j, \vec{b}_2) have the same atomic type due to the Substructure Lemma. Furthermore, by Ramsey's Theorem for structures, one can find i_1, i_2, i_3 such that $(a_{i_1}, a_{i_2}, \vec{b}_1)$ and $(a_{i_1}, a_{i_3}, \vec{b}_2)$ have the same atomic type. But then $\mathcal{T}_1 \stackrel{\text{def}}{=} \mathcal{S}' \upharpoonright \{a_{i_1}, a_{i_2}, b_1, \dots, b_{n'}\} \simeq \mathcal{S}' \upharpoonright \{a_{i_1}, a_{i_3}, b_1, \dots, b_{n'}\}$ via the isomorphism that maps a_{i_1} and each b_i to itself and a_{i_2} to a_{i_3} .

◀

4.2 Maintaining ECRPQs

Here we study the maintenance of ECRPQs and provide results in restricted settings. First we show that answers to an ECRPQ can be maintained in DYNFO on acyclic graphs. Even more, answers to the following extension of ECRPQs introduced in [4] can still be maintained. An ECRPQ with *linear constraints on the number of occurrences of symbols* on paths over an alphabet $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ is of the form

$$\mathcal{Q}(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} R_j(\vec{\omega}_j), A\vec{\ell} \geq \vec{b}$$

where $A \in \mathbb{Z}^{h \times (km)}$ for some $h \in \mathbb{N}$, $\vec{b} \in \mathbb{Z}^h$, and $\vec{\ell} = (\ell_{1,1}, \dots, \ell_{1,k}, \dots, \ell_{m,1}, \dots, \ell_{m,k})$. The semantics extends the semantics of ECRPQs as follows: for each $1 \leq i \leq m$ and $1 \leq j \leq k$, the variable $\ell_{i,j}$ is interpreted as the number of occurrences of the symbol σ_j in the path π_i . The last clause of the query \mathcal{Q} is true if $A\vec{\ell} \geq \vec{b}$ under this interpretation.

► **Theorem 16.** *Every ECRPQ with linear constraints on the number of occurrences of symbols is maintainable in DYNFO on acyclic graphs.*

Proof. Let $\Sigma = \{\sigma_1, \dots, \sigma_k\}$. We show how to maintain the answer of an ECRPQ \mathcal{Q} with linear constraints with only one regular relation R on an acyclic Σ -labeled graph $G = (V, E)$. Thus \mathcal{Q} is of the form:

$$\mathcal{Q}(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), R(\pi_1, \dots, \pi_m), A\vec{\ell} \geq \vec{b}$$

An arbitrary ECRPQ with linear constraints can be rewritten in this form by using closure properties of regular relations.

In a first step we reduce this problem to a structurally simpler one: the problem of maintaining \mathcal{Q} on a Σ -labeled graph consisting of m disjoint acyclic graphs G_1, \dots, G_m , restricted in such a way that solutions may only map the variables x_i, y_i to nodes in G_i , for each $1 \leq i \leq m$. The simple reduction from the original problem copies the queried graph m times. As m is a constant, this is a bounded first-order reduction.

Let $\mathcal{A} = (Q, (\Sigma \cup \perp)^m, \delta, s, F)$ be a finite automaton with padding symbol $\perp \notin \Sigma$ that recognizes the m -ary regular relation R . The idea is to maintain $(2m + km)$ -ary auxiliary relations $R_{p,q}$ for all $p, q \in Q$ intended to store a tuple $(\vec{x}, \vec{y}, \vec{\ell}_1, \dots, \vec{\ell}_m)$ with $\vec{x} = (x_1, \dots, x_m)$, $\vec{y} = (y_1, \dots, y_m)$ and $\vec{\ell}_i = (\ell_{i,1}, \dots, \ell_{i,k})$ if and only if the state q is reachable from the state p in \mathcal{A} by reading a tuple of words $(\lambda(\rho_1), \dots, \lambda(\rho_m))$, where for each $1 \leq i \leq m$, ρ_i is a path in G_i from x_i to y_i , and $\ell_{i,1}, \dots, \ell_{i,k}$ are the number of occurrences of the symbols $\sigma_1, \dots, \sigma_k$ in the label sequence of ρ_i .

We show how to express the query relation Q by these relations. To this end observe that the (fixed) linear inequality system $A\vec{\ell} \geq \vec{b}$ can be defined by a $(m \times k)$ -ary first-order formula $\psi_{A,\vec{b}}(\vec{\ell}_1, \dots, \vec{\ell}_m)$ that uses the built-in arithmetic.

The query relation Q is then defined by the following formula:

$$\varphi(\vec{z}) \stackrel{\text{def}}{=} \exists \vec{v} \exists \vec{\ell}_1 \dots \exists \vec{\ell}_m \bigvee_{f \in F} R_{s,f}(\vec{x}, \vec{y}, \vec{\ell}_1, \dots, \vec{\ell}_m) \wedge \psi_{A,\vec{b}}(\vec{\ell}_1, \dots, \vec{\ell}_m)$$

Here the existentially quantified variables \vec{v} correspond to variables of \mathcal{Q} that do not occur in the head of the query, and all x_i and y_i occur in either \vec{z} or \vec{v} .

The update formulas for the relations $R_{p,q}$ are similar in spirit to those for reachability in acyclic graphs used in Example 3. Suppose an edge (u, σ, v) is inserted into the graph G_i for some $i \in \{1, \dots, m\}$. The update formulas compose runs of \mathcal{A} from the runs stored in the relations $R_{p,q}$ as follows. For all states $p, q \in Q$, a tuple $(\vec{x}, \vec{y}, \vec{\ell}_1, \dots, \vec{\ell}_k)$ shall be in $R_{p,q}$ after the insertion if and only if it was in $R_{p,q}$ before the insertion or if the following conditions are satisfied:

- (a) There is a state $p' \in Q$, a tuple of nodes $\vec{x}' = (x'_1, \dots, x'_m)$ with $x'_i = u$, and vectors $\vec{a}_1, \dots, \vec{a}_k \in \mathbb{N}^m$, such that $(\vec{x}, \vec{x}', \vec{a}_1, \dots, \vec{a}_k) \in R_{p,p'}$.
- (b) There is a state $q' \in Q$, a tuple of nodes $\vec{y}' = (y'_1, \dots, y'_m)$ with $y'_i = v$, and vectors $\vec{b}_1, \dots, \vec{b}_k \in \mathbb{N}^m$, such that $(\vec{y}', \vec{y}, \vec{b}_1, \dots, \vec{b}_k) \in R_{p',q}$.
- (c) There is a tuple of symbols $\vec{s} \in (\Sigma \cup \perp)^m$ such that
 - (i) $s_i = \sigma$,
 - (ii) $s_j = \perp$ for each $j \neq i$ with $x'_j = y'_j$, and
 - (iii) there is an edge $(x'_j, s_j, y'_j) \in E_j$ for each $j \neq i$ with $x'_j \neq y'_j$
 and \mathcal{A} has a transition from p' to q' by reading \vec{s} .
- (d) $\vec{\ell}_j = \vec{a}_j + \vec{b}_j + \vec{c}_{\sigma_j}$ for each $j \in \{1, \dots, k\}$, where $\vec{c}_{\sigma_j} \in \{0, 1\}^m$ is the vector whose r th component is 1 if the r th component of \vec{s} is σ_j , and 0 otherwise.
- (e) $A\vec{\ell} \geq \vec{b}$, where $\vec{\ell}$ is the concatenation of $\vec{\ell}_1, \dots, \vec{\ell}_k$.

The conditions (a)-(c) can be easily expressed by first-order formulas using existential quantification. The conditions (d)-(e) can be expressed by using built-in arithmetic: since the graphs G_1, \dots, G_k are acyclic, it is easy to see that numbers used in those conditions are polynomial in the size of the active domain. We can therefore build the needed arithmetic incrementally by Proposition 4.

Deletions can be handled along the same lines by using the technique from Example 3. ◀

It remains open whether the answer relation of ECRPQs can be maintained on general graphs, even when only insertions are allowed. Yet when the rational relations are restricted to be unary, the ECRPQs can be maintained under insertions. More formally, a *CRPQ with linear constraints on the number of occurrences of symbols* over $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ is of the form

$$\mathcal{Q}(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq m} L_j(\pi_j), A\vec{\ell} \geq \vec{b}$$

where L_j is a unary rational relation (that is, a regular language), and A , \vec{b} and $\vec{\ell}$ are as in the definition of ECRPQs with linear constraints.

► **Theorem 17.** *Every CRPQ with linear constraints on the number of occurrences of symbols is maintainable in DYNFO under insertions.*

Proof. Let $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ and \mathcal{Q} be a CRPQ over Σ with linear constraints on the number of occurrences of symbols as above. Further let $\mathcal{A}_j = (Q_j, \Sigma, \delta_j, s_j, F_j)$, $1 \leq j \leq m$, be finite state automata for the regular languages L_j occurring in \mathcal{Q} .

We exhibit a DYNFO-program with built-in arithmetic for maintaining \mathcal{Q} on general graphs under insertions. The necessity for built-in arithmetic can be removed by Proposition 4.

The idea is similar to the proof of the previous Theorem 16. We maintain $(k+2)$ -ary auxiliary relations $R_{p,q}^j$ for each $j \in \{1, \dots, m\}$ and all $p, q \in Q_j$ with the intention that $R_{p,q}^j$ stores a tuple $(x, y, \ell_1, \dots, \ell_k)$ if and only if the state q is reachable from state p in the automaton \mathcal{A}_j by reading the label of a path ρ between x and y in G such that ℓ_1, \dots, ℓ_k are the number of occurrences of $\sigma_1, \dots, \sigma_k$ in ρ .

Before sketching how to maintain the relations $R_{p,q}^j$, we show how they can be used to express the answer of \mathcal{Q} . As in the proof of Theorem 16 the (fixed) linear inequality system $A\vec{\ell} \geq \vec{b}$ can be defined by a $(m \times k)$ -ary first-order formula $\psi_{A,\vec{b}}(\ell_{1,1}, \dots, \ell_{m,k})$ that uses the built-in arithmetic. Then a tuple \vec{u} of nodes in G is in the answer of \mathcal{Q} if and only if the following formula holds:

$$\varphi(\vec{z}) \stackrel{\text{def}}{=} \exists \vec{v} \exists \ell_{1,1}, \dots, \ell_{m,k}, \bigwedge_{1 \leq j \leq m} \left(\bigvee_{f \in F_j} R_{s_j, f}^j(x_j, y_j, \ell_{j,1}, \dots, \ell_{j,k}) \right) \wedge \psi_{A,\vec{b}}(\ell_{1,1}, \dots, \ell_{m,k})$$

Here the existentially quantified variables \vec{v} correspond to variables of \mathcal{Q} that do not occur in the head of the query, and all x_j and y_j occur in either \vec{v} or \vec{z} .

A small technical issue arises from the fact that it is not obvious why the length of paths ρ_1, \dots, ρ_m witnessing that a tuple of nodes \vec{u} is in the answer of \mathcal{Q} is polynomially bounded. This, however, is necessary for being able to quantify the length $\ell_{1,1}, \dots, \ell_{m,k}$ and to use the built-in arithmetic for computations. Fortunately the length of (shortest) witness paths can be bounded by a fixed polynomial in the size of the active domain. This has been shown even for ECRPQs with such linear constraints in [4, Lemma 8.6].

Now we show how to maintain the relations $R_{p,q}^j$. The following notion is useful. A relation R stores the *Parikh distances* of a Σ -labeled graph if it contains a tuple $(x, y, \ell_1, \dots, \ell_k)$ if and only if there is a path ρ between x and y such that its label $\lambda(\rho_i)$ contains ℓ_i occurrences of the symbol σ_i for each $1 \leq i \leq m$. We observe that the relations $R_{p,q}^j$ can be defined from the Parikh distance relations of the product graphs $G \times \mathcal{A}_j$. Since the automata \mathcal{A}_j are fixed, a modification of G yields a bounded number of first-order definable modifications to $G \times \mathcal{A}_j$.

Thus in order to maintain $R_{p,q}^j$, it suffices to be able to maintain the Parikh distance relation of a Σ -labeled graph under insertions. However, the dynamic program for maintaining distances under insertions from Theorem 12 can be easily generalized to maintain Parikh distances. For the sake of completeness we present the general construction. The goal is to maintain an auxiliary relation S intended to store a tuple $(x, y, t, \vec{\ell})$ with $\vec{\ell} \stackrel{\text{def}}{=} (\ell_1, \dots, \ell_k)$ if there are (not necessarily distinct) paths ρ_1, \dots, ρ_t from x to y in G such that each symbol $\sigma_i \in \Sigma$ appears exactly ℓ_i times among all ρ_1, \dots, ρ_t paths. The update formula for S after

inserting an edge (u, σ, v) is as follows:

$$\begin{aligned} \phi_{\text{INS}_{E\sigma_i}}^S(u, v; x, y, t, \vec{\ell}) \stackrel{\text{def}}{=} & \exists t_- \exists t_+ \exists t_{\odot} \exists \vec{\ell}_- \exists \vec{\ell}_{+1} \exists \vec{\ell}_{+2} \exists \vec{\ell}_{\odot} \left(A(x, y, t_-, \vec{\ell}_-) \right. \\ & \wedge A(x, u, t_+, \vec{\ell}_{+1}) \wedge A(v, y, t_+, \vec{\ell}_{+2}) \\ & \wedge A(v, u, t_{\odot}, \vec{\ell}_{\odot}) \wedge (t_+ = 0 \rightarrow t_{\odot} = 0) \\ & \left. \wedge t_- + t_+ = t \wedge \vec{\ell}_- + \vec{\ell}_{+1} + \vec{\ell}_{+2} + \vec{\ell}_{\odot} + (t_+ + t_{\odot})\vec{e}_i = \vec{\ell} \right) \end{aligned}$$

Here, for clarity we quantify k -ary tuples of variables. The tuple \vec{e}_i contains zeroes except for its i -th component, which is 1.

The correctness of this update formula follows immediately from the proof of Theorem 12. \blacktriangleleft

We remark that already boolean ECRPQs cannot be maintained under insertions in DYNPROP due to lower bounds for non-regular languages [11], and boolean CRPQs with $k + 2$ existentially quantified node variables cannot be maintained in DYNPROP with k -ary relations due to a lower bound for the k -clique query [24].

5 Maintaining Reachability in Product Graphs

In this final section we study the reachability query for product graphs. In addition to its importance for the evaluation of fixed graph queries, reachability in graph products can be used to maintain the result of regular path queries in combined complexity (i.e., when the query is subject to modifications as well). Furthermore it is relevant in model checking, where subsystems correspond to factors in product graphs (see, e.g., [3]).

The results for maintaining all distances obtained in the previous section immediately transfer to reachability in simple graph products (see the discussion at the end of Section 2). A small technical obstacle arises from the fact that the reachability query does not come with built-in arithmetic, while the distance query studied so far does. However, this is not a problem due to Proposition 4.

► **Theorem 18.** *Let \mathcal{G} be a class of graphs and $m \in \mathbb{N}$. If all distances up to n^m on \mathcal{G} can be maintained in DYNFO with built-in arithmetic, then reachability in the product of m \mathcal{G} -graphs is maintainable in DYNFO (without built-in arithmetic).*

Proof. For DYNFO with arithmetic this follows from Fact 5. As reachability is a domain independent query the result follows from Proposition 4. \blacktriangleleft

Shortest paths in products of acyclic and undirected graphs are of length at most n and n^2 , respectively. For these two classes of graphs, reachability can therefore be maintained in products of polynomially many factors using the program for all distances. More precisely, this is doable for reachability between two specified nodes \vec{s} and \vec{t} as opposed to all pairs of nodes (as there are exponentially many nodes in such product graphs).

For directed graphs, shortest paths in products of polynomially many graphs can be of exponential length. For this reason, the approach to maintain reachability in such products via distances fails. Even more, it is unlikely that there is a DYNFO-program for this problem: it could be used to decide reachability in the product of polynomially many graphs in PTIME, which is NP-hard. This follows from a reduction from emptiness of intersections of unary regular expressions which is known to be NP-hard [10].

► **Corollary 19.** *Reachability can be maintained in DYNFO in the product of*

- (a) *polynomially many undirected graphs,*
- (b) *polynomially many acyclic graphs, and*
- (c) *a constant number of directed graphs under insertions.*

This follows immediately from Theorem 18, Theorem 13 and Theorem 12. Reachability in products of an undirected and an acyclic graph and similar constellations can, of course, also be maintained.

For labeled graph products, the following corollary follows immediately from the proof of Theorem 16.

► **Corollary 20.** *Reachability in products of constantly many acyclic Σ -labeled graphs can be maintained in DYNFO.*

In the following we generalize Corollary 19 to a broader class of graph products. In the product graphs considered so far, there is an edge from a node (x_1, \dots, x_m) to a node (y_1, \dots, y_m) if there is an edge (x_i, y_i) in every factor G_i . This can be seen as a completely synchronized traversal through the given graphs. The graph products to be introduced next allow for more flexible, partially synchronized traversals.

Let $(G_i)_{1 \leq i \leq m}$ be a sequence of graphs with $G_i \stackrel{\text{def}}{=} (V_i, E_i)$, and let $A \stackrel{\text{def}}{=} (\vec{a}_1, \dots, \vec{a}_k)$ be a list of tuples from $\{0, 1\}^m$, called *transition rules*. We often identify A with the matrix that has the tuples \vec{a}_i as columns. The *generalized graph product of $(G_i)_i$ with respect to A* , denoted $\prod_i^A G_i$, has nodes $V_1 \times \dots \times V_m$ and edges (\vec{x}, \vec{y}) defined by the first-order formula

$$\bigvee_{\substack{\vec{a} \in A \\ \vec{a} = (a_1, \dots, a_m)}} \bigwedge_{a_i=0} x_i = y_i \wedge \bigwedge_{a_i=1} E_i(x_i, y_i)$$

For example, the usual product of two graphs is defined by the transition rule $\{(1, 1)\}$, and the so called *cartesian product* is defined by the rules $\{(1, 0), (0, 1)\}$. We remark that generalized graph products have also been called *non-complete extended p-sums*, short: NEPS (see, for example, [19]).

► **Theorem 21.** *Reachability in generalized product graphs is maintainable in DYNFO under modifications to factors and transitions rules⁴ for*

- (a) *a constant number of directed graphs under insertions and a constant number of transition rules,*
- (b) *polynomially many acyclic graphs and a constant number of transition rules,*
- (c) *polynomially many undirected graphs and polynomially many transition rules.*

Proof sketch. As usual we assume built-in arithmetic, which can be removed by Proposition 4.

For (a) and (b), the key observation is that reachability in generalized graph products can be reduced to finding a solution in natural numbers to a linear equation system. Let $(G_i)_{1 \leq i \leq m}$ be a list of graphs, $\vec{x} = (x_1, \dots, x_m)$ and $\vec{y} = (y_1, \dots, y_m)$ nodes of $\prod_i^A G_i$, and let

$$D \stackrel{\text{def}}{=} \{\vec{d} = (d_1, \dots, d_m) \mid \text{there is a path from } x_i \text{ to } y_i \text{ in } G_i \text{ of length } d_i, \text{ for each } 1 \leq i \leq m\}.$$

Then there is a path from \vec{x} to \vec{y} in $\prod_i^A G_i$ if and only if there is a tuple $\vec{d} \in D$ and $n_1, \dots, n_k \in \mathbb{N}$ such that $n_1 \vec{a}_1 + \dots + n_k \vec{a}_k = \vec{d}$. A shortest path witnessing that two tuples

⁴ We permit single bit modifications to A , that is, modifying one bit of a transition rule at a time.

\vec{x} and \vec{y} are connected in a generalized product of constantly many directed graphs (or of polynomially many acyclic graphs) can be of at most polynomial length. In particular, we can restrict numbers $n_1, \dots, n_k \in \mathbb{N}$ to be of polynomial size.

The dynamic program for maintaining reachability in those graph products works as follows. It maintains all distances for each of the factors. Upon modification of a graph G_i , the program updates all distances for G_i (using the program for maintaining all distances). Then it guesses n_1, \dots, n_k by using existential quantification, computes $\vec{d} \stackrel{\text{def}}{=} n_1 \vec{a}_1 + \dots + n_k \vec{a}_k$, and checks for each component d_i of \vec{d} that in G_i there is a path from x_i to y_i of length d_i . Modifications of the transition rule are handled in a similar way.

For (c) we rely on the following fact, which is a consequence of the proof of Theorem 2 in [19].

► **Fact.** Let $G \stackrel{\text{def}}{=} \prod_i^A G_i$ be the generalized product of the undirected graphs $(G_i)_{1 \leq i \leq m}$ with respect to a list A of k transition rules. Let $\vec{x} \stackrel{\text{def}}{=} (x_1, \dots, x_m)$ and $\vec{y} = (y_1, \dots, y_m)$ be two nodes of G , let C_i be the connected component of x_i in G_i and assume that $C_{i_1}, \dots, C_{i_\ell}$ are the only bipartite components. Then there is a path from \vec{x} to \vec{y} in G if and only if

- for each $i \in \{1, \dots, m\}$ there is a path from x_i to y_i in G_i , and
- the linear equation system $B\vec{x} = \vec{d}$ is solvable over \mathbb{Z}_2 where
 - B is obtained from A by setting rows $r \notin \{i_1, \dots, i_\ell\}$ to zero, and
 - the r th component of $\vec{d} \in \mathbb{Z}_2^\ell$ is the parity of the distances between x_r and y_r for $r \in \{i_1, \dots, i_\ell\}$ and zero for $r \notin \{i_1, \dots, i_\ell\}$.

Note that since the component of x_r with $r \in \{i_1, \dots, i_\ell\}$ is bipartite, all paths between x_r and y_r have the same parity.

We use the above fact to construct a DYNFO-program that maintains whether there is a path from \vec{x} to \vec{y} in the generalized product of polynomially many undirected graphs $(G_i)_i$ with respect to polynomially many transition rules A under single edge modifications to factors and single bit modifications to transition rules.

The dynamic program maintains auxiliary data that contains (1) all distances for each of the factors (and thus, in particular, also whether there is a path from x_i to y_i and whether the component C_i that contains x_i is bipartite) and (2) whether the equation system $B\vec{x} = \vec{d}$ has a solution over \mathbb{Z}_2 . For the latter the program maintains whether $\text{rank}(B) = \text{rank}(B, \vec{d})$ over \mathbb{Z}_2 .

It is known that the rank of matrices can be maintained in DYNFO [6]. Even more, as observed by William Hesse, the algorithm from [6] can maintain the rank even when whole rows may be replaced.

Upon modification of a factor G_i , the program updates all distances for G_i using the dynamic program for maintaining distances in undirected graphs. If the modification yields a bipartite component C_i of G_i , then the i th row of B is replaced by the i th row of A and d_i is set to the parity of paths between x_i and y_i . If the component C_i became non-bipartite, then the i th row of B is replaced by the all-zero row of A and d_i is set to zero. On the other hand, if the i -th bit of transition rule \vec{a}_j in A is modified, then the i -th row of B is modified only at its j -th entry if and only if C_i is bipartite. In all scenarios, at most one row of B is modified. The program can therefore maintain the ranks of B and (B, \vec{d}) accordingly. Finally, if y_i is reachable from x_i in G_i for every $1 \leq i \leq m$ and $\text{rank}(B) = \text{rank}(B, \vec{d})$ then the query bit of the dynamic program is set true.

The update operations described above can be expressed by first-order formulas with the aforementioned auxiliary data.



Observe that deciding reachability in generalized products of (1) polynomially many graphs with constant many transition rules and of (2) polynomially many acyclic graphs with polynomially many transitions rules are NP-hard problems. More precisely, the first generalizes reachability in the product of polynomially many graphs, which we already discussed above. As for the second, notice that the problem of deciding the existence of a 0-1 solution of a linear equation $A\vec{x} = \vec{1}$, which is known to be NP-hard even for a 0-1 matrix A [5, Chapter 8], can be straightforwardly reduced to reachability in the generalized product of acyclic graphs when polynomially many transition rules are allowed (by using the distance and linear equations characterization used in the proof of Theorem 21). These problems are thus unlikely to be maintainable in DYNFO.

6 Conclusion

In this article we explored graph query languages in the dynamic descriptive complexity framework introduced independently by Dong, Su and Topor, and Patnaik and Immerman. Furthermore we investigated the strongly related question, under which conditions distances in graphs as well as reachability in product graphs can be maintained. Our work is only a first step towards a systematic understanding of graph queries in dynamic graph databases. In the following we discuss some interesting directions for further research.

For several restricted classes of graphs we exhibited first-order update programs for maintaining distances. We also showed that quantifier-free update formulas do not suffice. It remains open, whether distances can be maintained for general graphs; we conjecture that this is the case.

► **Open problem 1.** Exhibit a DYNFO-program for maintaining distances.

As we have seen, reachability in products of labeled graphs is related to maintaining fragments of the graph query language ECRPQ. While we showed that reachability can be maintained in labeled products of acyclic graphs, this problem is already much harder for products of undirected, labeled paths—not to mention arbitrary labeled graphs.

► **Open problem 2.** Find dynamic DYNFO-programs for maintaining reachability in products of restricted classes of labeled graphs.

Another interesting direction is to exhibit dynamic programs for other, more expressive query languages.

► **Open problem 3.** Identify further expressive query languages that can be maintained dynamically.

A candidate query language to be studied are nested regular expressions (NREs) [17]. NREs allow to express queries with some branching capabilities. For example, the NRE $(a[b])^*$ selects pairs of nodes that are connected by an a^* -labeled path such that every node on this path has an outgoing edge with label b . This query can easily be maintained in DYNFO, as it is bounded first-order reducible to reachability. On the other hand, it is already unclear whether the query $(a[bc])^*$ can be maintained in DYNFO.

References

- 1 Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1, 2008.

- 2 Pablo Barceló Baeza. Querying graph databases. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 175–188. ACM, 2013.
- 3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- 4 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31, 2012.
- 5 Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006.
- 6 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015.
- 7 Camil Demetrescu and Giuseppe F. Italiano. Maintaining dynamic matrices for fully dynamic transitive closure. *Algorithmica*, 51(4):387–427, 2008.
- 8 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In Catriel Beeri, Atsushi Ohori, and Dennis Shasha, editors, *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages, Manhattan, New York City, USA, 30 August - 1 September 1993*, Workshops in Computing, pages 295–308. Springer, 1993.
- 9 Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In Joachim Biskup and Richard Hull, editors, *Database Theory - ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings*, volume 646 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 1992.
- 10 Zvi Galil. Hierarchies of complete problems. *Acta Informatica*, 6(1):77–88, 1976.
- 11 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012.
- 12 Erich Grädel and Sebastian Siebertz. Dynamic definability. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 236–248. ACM, 2012.
- 13 William Hesse. The dynamic complexity of transitive closure is in DynTC0. *Theoretical Computer Science*, 296(3):473–485, 2003.
- 14 Peter Bro Miltersen. Cell probe complexity-a survey. In *19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1999.
- 15 Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. Dynamic graph queries. To be presented at ICDT 2016.
- 16 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.
- 17 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.
- 18 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008.
- 19 Dragan Stevanović. When is neps of graphs connected? *Linear Algebra and its Applications*, 301(1):137–144, 1999.
- 20 Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007.

- 21 Peter T Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.
- 22 Thomas Zeume. The dynamic descriptive complexity of k-clique. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 547–558. Springer, 2014.
- 23 Thomas Zeume. *Small Dynamic Complexity Classes*. PhD thesis, TU Dortmund University, 2015.
- 24 Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 38–49. OpenProceedings.org, 2014.
- 25 Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015.